EXAMPLE ATTACK DOCUMENTATION

Touch Screen Window Manager

Douglas W. Jones

Sept 26, 2005

Taxonomy: wholesale third-party firmware
Applicability: touch-screen voting systems using window managers

Method:

Many DRE voting systems use a window manager, frequently from Microsoft, but some open voting products will use the X window manager. On such systems, all display of text on the screen and interpretation of touches on the screen are generally done through window-manager routines. In many cases, the window manager is considered to be an industry-standard commercial off-the-shelf component, and is therefore subject to reduced scrutiny.

If the perpetrator can add code to the window manager, the behavior of the voting system can be modified in a way that alters the election outcome. For example, consider this attack that will favor candidates from the aaa party in states allowing straight party voting where the bbb party is the other major party and the ccc party is a strong third party:

Insert in the window manager code to detect that the current window includes the text "straight party", and that it includes the text "aaa", "bbb" and "ccc" in the same window. The window manager is programmed to misbehave whenever this combination is present in the window, but only on the first Tuesday after the first Monday of November, only when this window has been used at least 20 times, and only when the machine has been turned on for over 4 hours. The misbehavior is to misreport all touches in the vicinity of the text "ccc" as being in the vicinity of "aaa", thus stealing straight-party votes from the third party and giving them to the major party.

The code for this attack should of course be obfuscated, with misleading comments and carefully hidden function so that it evades the internal quality control checks of the software vendor. The art of obfuscated programming has been thoroughly explored.

There are, of course, many variations on this attack, some of which do not depend on the straight party option. For example, the attack can be limited to an office or it can apply broadly, throwing,

say, 10% of the third party vote to the favored party in all races.

Resource requirements: The perpetrator must have access to the source code of the window manager.

Potential gain:

The target should be around 1/3 of the straight party votes for a major third party. In the past 50 years, third parties have rarely earned over 5% of the vote, but sometimes up to 15% (George Wallace in 1968). The fraction of straight-party voters is hard to determine, but it will be significant only for parties that put up candidates for many different offices. In recent years, only the Greens and the Libertarians have managed this, so these are the natural third parties to attack. As a naive guess, it is unlikely that this attack would win more than 1% of the vote.

Likelihood of detection:

Because the attack code is embedded in third-party off-the-shelf software, it is unlikely to be subject to the same scrutiny as purpose-written voting code. Because it only manifests itself under conditions typical of real elections, it is unlikely to be seen in any testing by the commercial off-the-shelf vendor. The checks sensitive to the number of votes cast and the length of time the machine has been running will evade many pre-election tests and possibly even many ITA tests. Small additions to the conditions suggested above can make it evade ITA testing.

If a voter does notice that their vote was cast for the wrong candidate (and there are variations of this attack that evade detection by the voter) the problem can easily be blamed on the voter (you simply touched the wrong point on the screen) or on touch screen alignment.

Because the attack code is modest, stealing only a small fraction of the votes cast, it is unlikely to show up in post election audits.

Countermeasures:

Preventative measures:

Eliminate testing and source code inspection exemptions for inspection of third-party commercial off-the-shelf software.

Eliminate testing and source code inspection exemptions for emergency patches and bug fixes.

Eliminate dependencies on window-manager functionality from the voting application. Typically, this will involve "flattening" the code to eliminate deep hierarchies of reusable software components. Instead, the voting application should directly manipulate the display screen.

Eliminate text from the voting application. Instead, display all ballot content on the screen as images, with extremely dumb image display software used to place all voting-related text on the screen. It would be helpful if there were a guarantee that the system contained no OCR software that could examine images to detect embedded text (such software is becoming increasingly widely available as a software component and may soon become a standard off-the-shelf component for other software systems).

Eliminate access to the real-time clock, or alternatively, strictly audit all use of the real-time clock so that no use of the date, the time of day or the time since power-up is permitted except for the purpose of logging events in the system event log.

Detection measures:

Take voter complaints of the form "I voted straight party ccc and it marked the aaa candidate" very seriously. Unfortunately, variations on this attack may be invisible to the voter.

Perform parallel testing on election day, with a test environment that the machine cannot possibly distinguish from real use. The machine should be turned on at and off at reasonable times for polling places to be opened and closed, the number of votes should be typical of a busy polling place,

Citations:

The Fidlar and Chambers EV 2000 was accidentally "attacked" by Microsoft following a distant relative of this scenario in January 1998. The "attack" was a cosmetic change that involved no change to the Windows applications programmer interface (API) and was therefore determined exempt from testing by the ITA. Unfortunately, this cosmetic change ended up revealing, to each voter, all votes cast by the previous voter to use that machine. I described this to the House Science Committee on May 22, 2001. See http://www.cs.uiowa.edu/~jones/voting/congress.html

That accidental attack led me to propose this attack in *E-Voting -- Prospects and Problems*, April 13, 2000. Available on-line at http://www.cs.uiowa.edu/~jones/voting/taubate.html

Retrospective:

The problem posed by emergency security patches from vendors is extremely serious. These come with a built-in urgency that is immense. We are training a generation of computer system administrators to install such patches immediately and without question. It is not clear that this is prudent except when we know, with a great degree of certainty, that the vendors software development procedures conform to the same standards as our application.